RGG: Reactor Geometry (and Mesh) Generator

Rajeev Jain and Tim Tautges
Argonne National Laboratory
9700 S. Cass Avenue, Argonne, IL 60439
Tel: 630-252-3176, Fax: 630-252-5986, Email: jain@mcs.anl.gov, tautges@mcs.anl.gov

Abstract – *The reactor geometry (and mesh) generator RGG takes advantage of information about repeated structures in both assembly and core lattices to simplify the creation of geometry and mesh. It is released as open source software as a part of the MeshKit mesh generation library. The methodology operates in three stages. First, assembly geometry models of various types are generated by a tool called AssyGen. Next, the assembly model or models are meshed by using MeshKit tools or the CUBIT mesh generation toolkit, optionally based on a journal file output by AssyGen. After one or more assembly model meshes have been constructed, a tool called CoreGen uses a copy/move/merge process to arrange the model meshes into a core model. In this paper, we present the current state of tools and new features in RGG. We also discuss the parallel-enabled CoreGen, which in several cases achieves superlinear speedups since the problems fit in available RAM at higher processor counts. Several RGG applications—1/6 VHTR model, 1/4 PWR reactor core, and a full-core model for MONJU—are reported.*

I. INTRODUCTION

Creation of geometry and mesh are two important steps in the simulation of nuclear reactor cores. Such cores are typically formed by arranging pins in a lattice of surrounding material. A reactor can be described as a two-level hierarchy of lattices [1]. The first level corresponds to fuel or other assemblies consisting of cylindrical pins; in the second level, assemblies are arranged in a lattice to form the reactor core. Although the structure inherent in this two-level hierarchy could be used to automate parts of the generation process, experience shows that user interaction is often required. We describes a system for generating reactor core geometry and mesh models that balances lattice-guided automation and user interaction at key points in the process. This system, which we call RGG, for reactor geometry (and mesh) generator, can be formulated in a three-stage process. In the first stage, assembly geometry and meshing scripts are created; in the second stage the mesh for this assembly geometry is created; and in the third stage, the core model is created by using the output from the first two stages. RGG contains two tools, AssyGen and CoreGen, for modeling several types of nuclear reactor assembly and core models.

A literature review and various other domain-specific tools for geometry and mesh generation are discussed in an earlier paper [1]. During the development of the tools for RGG and motivated by discussions with neutronics and thermohydraulics groups, we added new features to the toolset, including support for tetrahedral meshing, creation of axially varying assemblies, and support for 2D core creation; these are reported in a recent journal publication [2]. In this paper, we focus on the parallel version of the tool, new models, and keywords for aiding postprocessing. The RGG toolset continues to grow more robust, with additional features and the ability to create large models automatically. Meshes generated by RGG tools can be used by solver codes that can use unstructured tetrahedral or hexahedral meshes, e.g. FE and SE (spectral element) codes. Specific examples include Star-CCM+, Star-CD, Fluent, Abaqus, etc., as well as codes developed at Argonne National Laboratory (ANL).

The paper is organized as follows. Section II describes the current status of tools and provides a brief overview. Section III describes the parallel version of the tool and new keywords to the input file language. Section IV describes core models created by using these tools and presents performance data from several applications. Section V discusses our conclusions.

II. ASSYGEN AND COREGEN

AssyGen and CoreGen are tools created as a part of MeshKit [3] library developed and maintained at ANL. These tools rely on geometry and mesh libraries developed as a part of Interoperable Tools for Advanced Petascale Simulations (ITAPS) project. The Common Geometry Module (CGM) [4] provides functions for constructing, modifying, and querying geometric models in solid model-based and other formats. Finite-element mesh and mesh-related data are stored in the Mesh-Oriented database (MOAB) [5]. MOAB also provides efficient functions for handling mesh in parallel. Mesh generation is performed by using a combination of tools. The CUBIT mesh generation toolkit [6] provides algorithms for both tetrahedral and hexahedral mesh generation. MeshKit provides efficient algorithms for mesh copy/move/merge, extrude, and other algorithms. The AssyGen tool generates the assemblies, and the CoreGen tool provides functions to copy/move/merge those assemblies to form a core.

AssyGen can generate rectangular or hexagonal assemblies. Figure 1 shows two such hexagonal assembly geometries being created. It highlights the first two stages of the process. Input to AssyGen tool is a keyword-based text file. Output is assembly geometry and a mesh script.

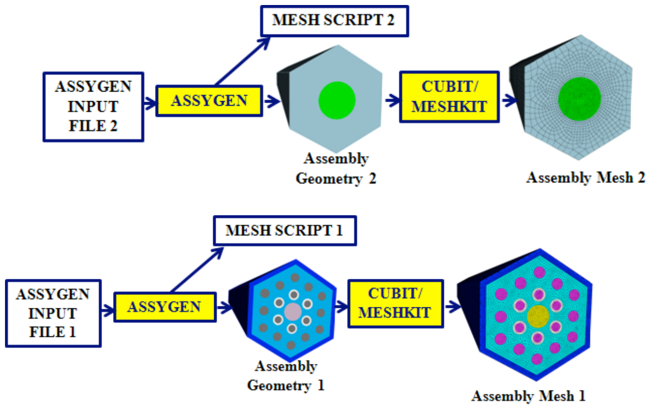

Fig. 1. First two stages of the geometry/mesh process, where AssyGen and CUBIT are executed for each assembly type.

The input file is based on a predefined set of keywords, followed by values that describe the model. A complete list of keywords, options, and values is given in the README file [7]. The geometry file created by AssyGen can be saved in formats supported by the geometry engine used to build CGM. ACIS [8] and OpenCascade [9] are currently supported by CGM. In the input file, we describe an assembly as a lattice of unit cells. Each unit cell has zero or more concentric cylindrical layers of material, cut from a background material defined for each unit cell or for the assembly as a whole. Unit cell shapes can also be imprinted on the background material, to more finely control the mesh in each unit cell. The assembly can be surrounded by one or more layers of duct wall material. Multiple pin-cell types can be defined, each with one or more concentric cylinders of material and a background material for the cell. Cylinders input for individual pin cells can be larger than the unit cell dimensions; these volumes will overlap neighboring pin-cell regions when present. In this case, a special keyword can be used to restrict these larger structures to the unit cell so that they do not overlap neighboring regions. Empty pin cells can be specified in the assembly lattice by a predefined "XX" or NULL unit cell type, indicating that only background material (and structures from neighboring unit cells) overlaps the cell. Parameters can be specified multiple times with varying Z-dimensions and material properties to create assembly models with axially varying properties.

In the second stage of the process, assembly geometry and the CUBIT mesh script, which is automatically generated by AssyGen, are run to generate an assembly mesh. This assembly mesh has the materials and boundary conditions defined. Top, bottom, and side surfaces of all materials are marked as boundaries; and material names are suffixed with "_top," "_bot," and "_side" to name the boundary conditions, respectively. Material names are defined in the text-based input file. AssyGen can create both surface- and volume-type geometry files and their corresponding mesh script files.

We note that for a given core configuration, the first two stages are repeated for each assembly that forms the core (see Fig. 2 for the core formed by assemblies created in Fig. 1). The user must ensure that the meshes match between assemblies. The tools do not explicitly enforce a constraint to match the nodes along the sides of the assemblies that sit next to each other. There are keywords in AssyGen to assign the intervals along the edges and in the z-direction. The same interval or bias factor must be used on all the assemblies to guarantee that neighboring assemblies are glued perfectly and the resulting core mesh is conformal. For a tetrahedral mesh, enforcing this constraint is difficult. Both translational and rotational symmetry of meshes on the side surface of all the assemblies are desired. This was achieved by meshing the sides first and then meshing the entire volume. The side surfaces are split and have the same mesh interval on the sides. Half of the split is meshed and then flipped onto the other half-surface [2].

The meshing process is brittle and often sensitive to the input mesh size. This problem occurs for several reasons. The top surface is cut by a large number of cylindrical rods and is often hard to mesh by using unstructured quadrilateral elements. Also, the ratio of largest to smallest dimension on the top surface of the model is large, which requires having a very small element size to mesh the top surface. The tight element budget and specific needs of the simulation scientists make it clear that finer control and freedom are required during this stage of the reactor core generation process. At present we use CUBIT for the mesh generation process; specific keywords such as EdgeInterval, RadialMeshSize, and AxialMeshSize are available as variables in the script to better control the meshing operation. Since CUBIT is closed-source, efforts are being made to develop algorithms such as Jaal [10] in the MeshKit library to tackle the mesh generation problems.

CoreGen reads all the assembly mesh files and a keyword-based text input file similar to the AssyGen input file. This file describes the arrangement of assemblies in the core lattice and locations of models for each assembly type, along with those meshes; this information is used to generate the overall core mesh. Figure 2 shows two hexagonal assembly mesh files, an interstices mesh, and a CoreGen input file. These files are read by the CoreGen program to create a makefile and a full hexagonal core mesh file with 19 assemblies according to the specification in the CoreGen input file. Unlike the example in Fig. 2, which creates a core mesh, CoreGen can read in geometry files to create the resulting core geometry, in which case the overall process becomes a two-stage process with no meshing. The interstices mesh is not copied/moved as are the other assembly mesh files; it is a way to provide fixed pieces in the model. The makefile generated by CoreGen automates the whole process, from various assembly to the creation of the core. Figure 3 in Section IIIA shows the generation of the same core mesh using parallel-enabled CoreGen.

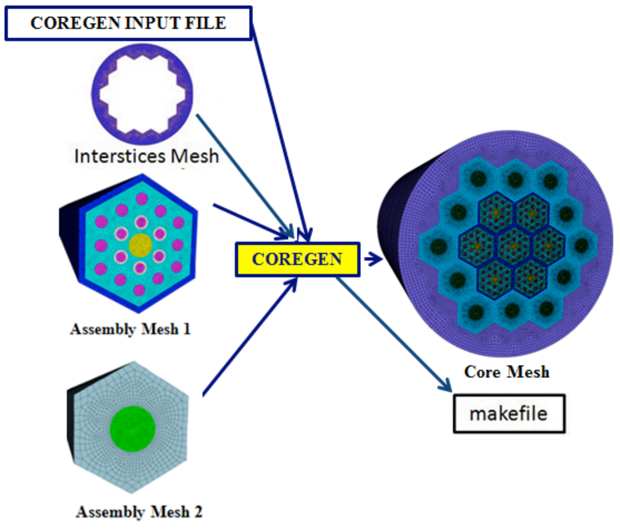

Fig. 2. Third stage of the geometry/mesh process, where CoreGen is executed.

CoreGen supports creation of 1/6, 1/12, and full-core models for a hexagonal-type core and a full-core model for a rectangular-type core. Details on the specifications and conventions used for core definition can be found in [1]. The examples listed in Section - IVA involves a 1/6 hexagonal core, IVB involves a full hexagonal core and IVC involves a full rectangular core. The example in Section IVA also highlights another variation of this three-stage process. A 2D core is formed in the first three stages; in the third stage, according to the user specification, the 2D core model is extruded to the desired height and number of subdivisions to create a 3D core mesh. This 2D core generation plus extrusion process is faster than its 3D counterpart.

We note that metadata, or material and boundary conditions, must propagate from individual assemblies to the core. To this end, abstractions are defined specifying the handling of specific types of groupings [2]. The groupings are of three types: copy, expand, and extrude. When an entity is copied and new entities are created, it is assigned to a copy grouping. When the grouping needs to accommodate and expand, it is assigned to an expand grouping. For extrude grouping, the entities of a group are replaced by the newly created entities of the corresponding higher dimension that were extruded.

Often, the sides or faces of an overall core model are desired as boundary conditions. We use the NeumannSet keyword in the input file for top, bottom, and side faces; for side faces, the equation of line in radial direction must be specified along with the keyword. This radial line sets the particular face or side of the core for which the boundary condition is desired. Internally these boundary conditions on the core are assigned by obtaining the skin of the entire core and filtering the faces into relevant groups specified in the input file.

III. NEW SALIENT FEATURES

For several models the mesh size did not fit in the available memory, and we realized a need for a parallel-enabled CoreGen. This parallel version is described in Section IIIA. In Section IIIB, we describe new options and the Info keyword, which helps keep track of pin and assembly numbers in both the assembly and core mesh files.

IIIA. Parallel-Enabled CoreGen

During creation of core models with large numbers of assemblies formed with only a few types of specific assemblies, we realized that the large memory requirement was a bottleneck. Recent development in the parallel capabilities for handling and manipulating meshes in MOAB created a perfect environment for a parallel-enabled CoreGen. CoreGen itself is ideally suited for parallelism. The basic algorithm used for parallelizing can be summarized in five steps:

1. On each processor: read CoreGen input file, parse, and determine assembly copies assigned to this processor based on a round-robin distribution.
2. Locally, on each processor, read assembly meshes for assemblies determined in step 1.
3. Perform assembly copy/move operations assigned to this processor.
4. Perform parallel merge.
5. Save output mesh.

For steps 4 and 5, CoreGen leverages the parallel merge and save algorithms developed in MOAB [5]. New algorithms were developed for shared vertex and metadata resolution among the processors [11].

The copy/move task distribution is deterministic; it is done on each processor based on the text-based CoreGen input file. In step 1, three cases arise when distributing the assembly meshes among processors:

A. np < nA
B. nA < np < nT
C. np > nT

For case A, each processors loads more than one assembly mesh file and solely performs the copy/move operation associated with that assembly for the entire core. For case B, some mesh files are loaded in multiple processors;. The mesh file selection is based on the frequency or the number of occurrence of that mesh file in the core. The file that appears most number of times in the core is assigned to multiple processors. This operation is deterministic and performed by all processors. For case C, some processors remain idle, and copy/move task is divided per assembly; therefore, only nT processors can take part in this parallel algorithm.

Figure 3 presents a simple example to explain the parallel algorithm. Four processors, P0 to P3, are used; individual assemblies are numbered from 1 to 19. The model uses two assembly mesh files and one interstices mesh file. Figure 3A shows the processor, the mesh file loaded, and the copy/move task. Figure 3B shows the CoreGen output mesh for a full hexagonal core with numbered assembly locations.

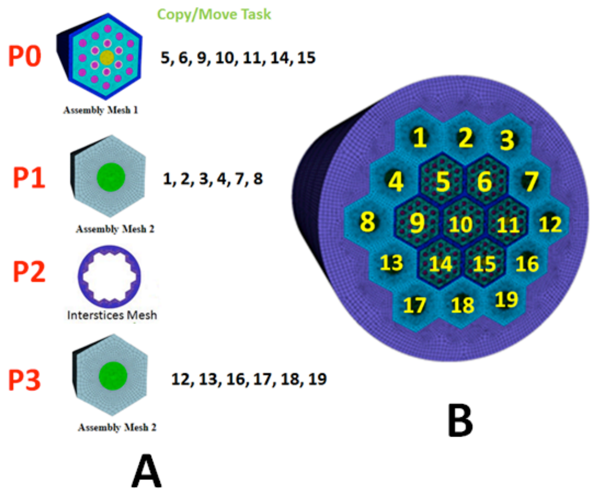

Fig. 3. Third stage of the geometry/mesh process, where CoreGen is executed.

Processor P0 loads assembly mesh 1, moves it to location 5, and then copies it to locations 6, 9, 10, 11, 14, and 15. Both processors P1 and P3 load assembly mesh 2; this mesh occurs 12 times in the core. The copy/move task is shared by P1 and P3, each handling 6 assemblies. P2 loads the interstices mesh and does not participate in the copy/move process. Once the copy/move task is complete, CoreGen performs a parallel merge. This algorithm does not delete mesh matching nodes; rather, the parallel sharing information is modified to indicate that they are the same logical vertex. 2D and 3D decomposition schemes for performing the parallel merge are implemented [11]. Parallel merge mesh also significantly lowers the total wall-clock time, as shown in Section IVA. The mesh then is saved in parallel by using MOAB's parallel HDF5-based writer. This writer can write a single output mesh file combining input from individual processors. Application codes may require having the core mesh in separate small files for each, or they may require one mesh file from all the processors. The CoreGen input file language defines a keyword SaveParallel that helps in specifying the option to save a mesh file from individual processors, one mesh file from all processors, or both.

IIIB. Keywords and Options

Two command line options, "-m" and "-j," are added to CoreGen and AssyGen, respectively. The "-m" option does not run the process; it only creates a makefile, which is often desired for automatically generating the assemblies forming the core. The "-j" option only creates a journal or mesh script file, without creating the geometry. An option "-t" was added to both the tools to print out detailed timing information. For CoreGen, the timing information is printed at the end of each step; for the parallel version, the maximum wall-clock and CPU times required by a processor for each subprocess are given.

Fuel and other coolant pins are grouped by materials in the resulting assembly and core mesh. One often wishes to keep track of the pin/assembly number in the assembly/core model, respectively. Postprocessing is an area where marking the pin and assembly numbers proves useful. For instance, the radiation or temperature profiles for a particular pin in a particular assembly are measured experimentally and must be validated by simulations; experience has shown that it is tedious to recognize and locate that pin in the hundreds of thousands of pins that form the reactor core. To avoid this problem, we create information files as output of the AssyGen and CoreGen programs. These files can be loaded into the simulation software and populated along with the mesh to label the pins and assemblies. The Info keyword was introduced in both AssyGen and CoreGen to trigger the tools to generate extra files specifying pin/assembly number and their location. Both AssyGen and CoreGen generate a file with the base name suffixed with "_info.csv." For AssyGen, this file contains the pin-cell number and location of the pin; for CoreGen the file contains the assembly number, assembly index (mesh files 1 to 2 and numbers 1 to 19 in Fig. 3), and the center of the assembly. CoreGen also creates a file with the base file name suffixed with "_mesh_info.csv". This contains the pin number and the centroid of each element that belong to the specified pincell. The convention for numbering of pins and assemblies is defined in our previous paper [1]. Internally in AssyGen during geometry creation each pin is assigned a NAME tag [3] for the material and an extra NAME tag for the pin number. We trick the system by creating each pin as a separate material or region in the AssyGen stage. In the CoreGen stage, after retrieving the pin number from the assembly regions, these extra materials are deleted.

The Info keyword technique has been demonstrated with STAR-CCM+, where after loading the simulation file, the info files with the pin/assembly number and cell centroids are used mark the pins. This is achieved by using annotations defined in STAR-CCM+ simulation interface. A Java macro was written to read the info file and populate the pin/assembly numbers based on the cell centroid onto the existing simulation; the macro can be initiated from STAR-CCM+ GUI or command line. For large models running this macro can be compute intensive, we are currently investigating this issue.

IV. EXAMPLES

In this section we present several models with focus on meshing and parallel-enabled CoreGen. Earlier papers present results from RGG generated meshes using simulation tools like UNIC [1], Star-CCM+ [2] and Abaqus [13].

IVA. Sixth-Core Model

For our first example, we construct the model shown in Fig. 4, one-sixth of the very high temperature reactor (VHTR) core. This example is available in the MeshKit repository [7]. We create this model using three techniques and compare the performance of each technique. The model consists of 11.8M hexahedral elements, 14M mesh vertices and a total of 58 (nT) assemblies. 12 (nA) different assemblies mesh files are required to describe this 58 assembly core. The model is tailored to automatically run and create the core mesh from scratch without user interaction. Appropriate values of axial and radial mesh size are specified in the input files. A makefile generated by CoreGen is used to run the problem.

First, this model was created by using the three-stage process described in this paper. The tools were run on a desktop Linux-based workstation with a clock speed of 2.5 GHz and 12 GB RAM. The three-stage process without extrusion took 4 minutes to generate the geometries using AssyGen, 5 minutes to create hexahedral assembly meshes using CUBIT (version 12.2), and 15 minutes to generate the core model using CoreGen. Thus, the total time using this method was 24 minutes.

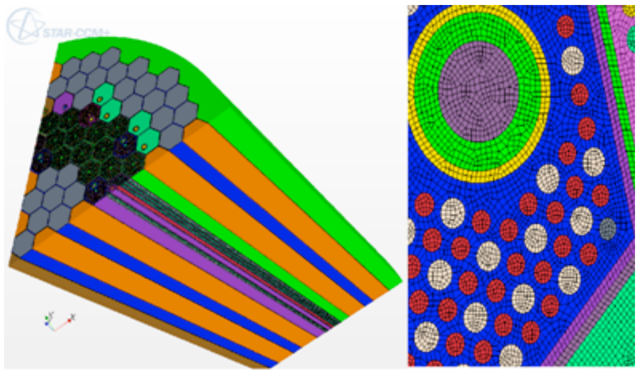

Fig. 4. One-sixth of a VHTR core model generated by using CoreGen (left); a closeup of assembly mesh in this model (right).

Second, the same model was constructed by using a four-stage process: (1) two-dimensional assemblies were created by using AssyGen; (2) CUBIT was used to mesh these assemblies (with quadrilaterals); (3) a two-dimensional core was generated by using CoreGen; and (4) the entire 2D core mesh was extruded into the third dimension. Using this extrusion-based approach required 0.6 times the execution time of the three-stage process. The total execution time was reduced from 24 minutes to 15 minutes, or by almost 40%. This reduction was due primarily to the reduced number of vertices that needed to be copied, moved, and merged between assemblies during the CoreGen stage. For example, the number of vertices considered for merging was reduced from 162,690 in the 3D process to only 9,570 in the 2D CoreGen process.

The minimum and maximum shape metrics for the VHTR mesh are 0.00125 and 0.00618, respectively. These metrics are low (normally, shape metric values above 0.2 are deemed acceptable). However, the metrics are due to the high aspect ratio of the assemblies (each assembly is 7.93 meters in length but only about 37 cm across) and the resulting mesh.

The performance of parallel CoreGen was measured by using the model without extrusion or the three-stage process model described above. When using 58 processors for running CoreGen, the total execution time for creating this model is less than 10 minutes (9 minutes of serial execution time: AssyGen and Meshing + 0.33 minutes of CoreGen time). CoreGen takes only 0.33 minutes, compared with 15 minutes in the serial case.

TABLE I

CPU time in minutes and maximum memory used for 1/6 VHTR core with 11.8M hexes

procs	Copy/Move (mins)	Merge (mins)	Save (mins)	Total (mins)	Memory (GB)
1	10.3	3.8	0.4	14.7	3.18
8	5.8	7.5	0.35	13.8	2.13
16	0.2	6.7	0.18	7.2	1.33
32	0.03	0.9	0.09	1.06	0.41
58	0.004	0.2	0.06	0.33	0.2

Table I lists the CPU time and the maximum memory used for various steps of the CoreGen stage, with different numbers of processors. As the table indicates, all the operations achieve superlinear speedups in some cases; memory usage data indicates that these steps are where the application goes from swapping to a state where the job fits in available memory. These results indicate one important reason for parallelizing RGG: so the application can fit in memory without swapping. Merging is observed to actually slow down going from one to eight processors; this slowing is probably due to the communication overhead required in the parallel algorithm. At larger numbers of processors, however, the merge time is reduced far below the serial time. As expected, the total time (time taken to save and maximum memory used by a processor) decreases with an increased number of processors.

IVB. Full-Core MONJU Reactor

Figures 5 and 6 show a full-core MONJU reactor, which comprises 8 assembly types and consists of 715 assemblies in total. AssyGen and meshing take 5.5 minutes (serial process), whereas CoreGen on 712 processors on the Fusion cluster at ANL takes only 1.5 minutes to copy/move/merge and save 8 assemblies to 715 different locations in the core. The total wall-clock time required to generate this 101M hexahedral element model is 7 minutes.

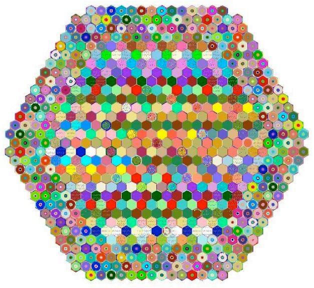

Fig. 5. Full-core MONJU reactor (top view).

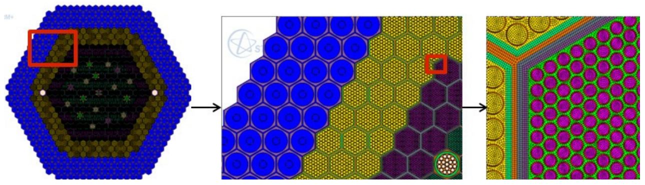

Fig. 6. Full-core MONU reactor; closeup area in red rectangular region is highlighted from left to right.

IVC. 1/4 PWR Core

Figure 7. shows the geometry of benchmark problem "MOX Fuel Loaded Small PWR Core"; a detailed description can be found on the website of Nuclear Reactor Analysis and Particle Transport Lab [12]. There are 10 (nA) different assembly mesh files; 3 UOX, 3 MOX and 5 baffle assemblies. Total number of assemblies in the model (nT) is 25. Individual assembly geometries are created by using the AssyGen tool. CoreGen then is used to copy/move the assemblies and form the core geometry. Note that trivial assemblies that do not contain any rods or the baffles are generated directly by using CUBIT. When creating the core model, the NeumannSet keyword is used to create individual side faces of the core as boundary conditions. Figures 7A, 7B, and 7C are closeups of the area in the red rectangular region highlighted on the core model. The model consists of approximately 11,000 volumes. On a Linux desktop, the assembly geometry creation takes 8 minutes; CoreGen takes 12 minutes of wall-clock time and uses 0.9 GB of RAM to create this core geometry.

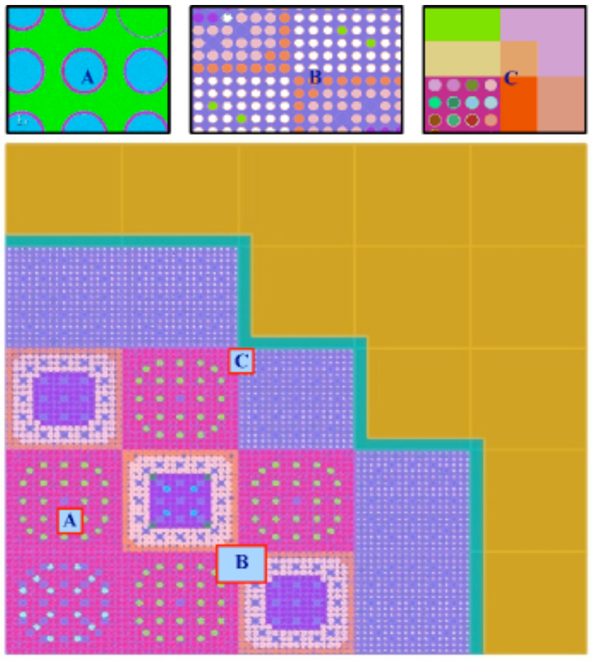

Fig. 7. 1/4 PWR benchmark geometry with closeup views A, B, and C showing details of the model.

Note that no mesh is generated and the core shown in Fig. 7 is a geometry generated by CoreGen. In Table II we list the performance metrics for the mesh generation of the same core. Individual assemblies geometries were created using AssyGen, meshing was performed using CUBIT and CoreGen was run in serial and parallel to create a core mesh file. . The resulting core mesh has 5.21 M elements, for each assembly the number of intervals in x and y direction is set to 67 and in z or axial direction it is 16. The disk space occupied by the resulting core mesh file is 0.78GB. Table II shows the values of maximum CPU time taken by individual components of the CoreGen process

TABLE II

CPU time in mins and maximum memory used for 1/4th PWR core with 5.2 M hexes.

procs	Copy/Move (mins)	Merge (mins)	Save (mins)	Total (mins)	Memory (GB)
1	0.94	0.01	0.24	2.2	1.65
4	0.33	9.1	0.78	10.54	1.12
8	0.27	6.05	0.5	7.75	0.93
16	0.07	0.04	0.43	1.33	0.33
25	0.01	0.01	0.18	0.51	0.21

The total number of processors for running parallel-enabled CoreGen is limited to 25 (nT), since there are 25 assemblies that form the core. There is an increase in merge and save time when using 4 and 8 processors, this is due to large communication cost and uneven distribution of meshes among processors which increases the total time required to create the model. For 16 and 25 processors the total time starts to decrease as the load is more balanced each processor loads exactly one mesh file, thus the parallel communication time for merge and save operation is lower than that for 4 and 8 processors. For all cases, the copy/move time and maximum memory required by a processor decrease, this behavior is due to sharing of copy/move task and distribution of meshes among processor compared to serial process. Unlike example IV A in this example speedup observed are not superlinear, because in this example the difference between nA and nT is not large and the parallel communication time at low processor count is high. Speedup seems to depend on nA (10) and nT (25) of the model. nT limits the number of processors that CoreGen is able to uses, also, a low value of nA causes the copy/move task to be distributed among the processors more evenly, since fewer component assemblies form the core.

V. CONCLUSION

The original three-stage approach for generating lattice-based models reported in earlier papers has proved useful. The CoreGen tool was modified to work in serial and parallel; parallelization is coarse-grained. Each processor that participates in the parallel algorithm must load atleast one of the assemblies forming the core. The parallel version allows the problem to fit in memory, thereby significantly reducing the total time and maximum memory per processor required for generating large models. The speedup for parallel-enabled CoreGen is problem and processor dependent. Usually better speedups are obtained when the number of processors is equal or more than the number of mesh files input to CoreGen. This effort has also has prompted new developments in parallel save and mesh merge algorithms. The AssyGen tool was enhanced with new keywords to aid creation of matching meshes between different assemblies forming the core. New options for reporting the timing, creation of only mesh script and makefile also have been added. Experience in using the tools has helped the development of an Info keyword, which generates new files along with model files; these info files contain the pin number, assembly number, and their location and can be useful in postprocessing and other such activities. Moreover, new types of reactors have been generated by using the tools. For example, the full-core MONJU reactor example demonstrates the power of the parallel-enabled CoreGen tool. Geometry-only models also can be created; the 1/4 PWR core geometry creation is presented as an illustration. Performance results for both 1/6 VHTR and 1/4 PWR model show savings in both maximum memory used per processor and total time required.

ACKNOWLEDGMENTS

We thank Jason Kraftcheck for his work on MOAB's parallel writer, Nathan Bertram for parallel merge mesh capabilities in MOAB, J. H. Thomas and W. David Pointer for helpful discussions and feedback during the development of the tools. We also thank the Fathom group at ANL, who maintain the libraries required by this tool. This work was supported in dpart by the U.S. Dept. of Energy Office of Nuclear Energy Nuclear Energy Advanced Modeling & Simulation (NEAMS) Program; by the U.S. Dept. of Energy Office of Advanced Scientific Computing Research, Office of Science; and by the U.S. Department of Energy's Scientific Discovery through Advanced Computing program, under Contract DE-AC02-06CH11357.

NOMENCLATURE

nA:	Number of different assemblies forming the core.
nP:	Number of processors.
nT:	Total number of assemblies forming the core.

REFERENCES

1. T. J. TAUTGES and R. JAIN, "Creating geometry and mesh models for nuclear reactor core geometries using a lattice hierarchy-based approach," in *Proc. 19th International Meshing Roundtable,* Springer-Verlag, Berlin, 351–366 (2010).

2. T. J. TAUTGES and R. JAIN, "Creating geometry and mesh models for nuclear reactor core geometries using a lattice hierarchy-based approach," *Engineering with Computers* (2011).

3. T. J. TAUTGES, J. KRAFTCHECK, J. PORTER, A. CACERES, I. GRINDEANU, D. KARPEEV, R. JAIN, H.-J. KIM, S. CAI, S. JACKSON, J. HU, B. SMITH, C. VERMA, S. SLATTERY, and P. WILSON, MeshKit: An open-source library for mesh generation," in *Proc. SIAM Conference on Computational Science & Engineering* (2011).

4. T. J. TAUTGES, "CGM: A geometry interface for mesh generation, analysis and other applications," *Engineering with Computers*, 17, 486–490 (2005).

5. T. J. TAUTGES, R. MEYERS, K. MERKLEY, C. STIMPSON, and C. ERNST, MOAB: A mesh-oriented database, *SAND2004-1592*, Sandia National Laboratories, Albuquerque, NM (2004).

6. G. D. SJAARDEMA, T. J. TAUTGES, T .J. WILSON, S. J. OWEN, T. D. BLACKER, W. J. BOHNHOFF, T. L. EDWARDS, J. R. HIPP, R. R. LOBER, and S. A. MITCHELL, *CUBIT mesh generation environment volume 1: Users manual*, Sandia National Laboratories, Albuquerque, NM (1994).

7. MeshKit website (2012), http://trac.mcs.anl.gov/projects/fathom/browser/

8. Spatial website (2012) http://www.spatial.com/

9. Open CASCADE Technology website (2012), http://www.opencascade.org.

10. C. S. VERMA and T. TAUTGES, "Jaal: Engineering a high quality all-quadrilateral mesh generator," in *Proc. 20th International Meshing Roundtable*, edited by W. R. Quadros, Springer (2011).

11. 11. T. J. TAUTGES, J. KRAFTCHECK, N. BERTRAM, V. SACHDEVA, and J. MAGERLEIN, "Mesh interface resolution and ghost exchange in a parallel mesh representation," *Presented at the 26th IEEE International Parallel & Distributed Processing Symposium*, Shanghai, China, May 21 (2012).

12. 1/4 PWR Benchmark Problem (2012), http://nurapt.kaist.ac.kr/benchmark/

13. SUBHASHISH MOHANTY, RAJEEV JAIN, SAURIN MAJUMDAR, TIMOTHY J. TAUTGES and SRINIVASAN MAKUTESWARA, "Coupled Thermal-Irradiation-Structural Analysis of HGTR Fuel Brick Using ABAQUS". *Paper number 12352, ICAPP 2012, Chicago, USA*, June (2012).